

## Getting More From Your Multicore: Exploiting OpenMP for Astronomy

Michael S. Noble

*Kavli Institute for Astrophysics, Massachusetts Institute of Technology*

**Abstract.** Motivated by the emergence of multicore architectures, and the reality that parallelism is rarely used for analysis in observational astronomy, we demonstrate how general users may employ tightly-coupled multiprocessors in scriptable research calculations while requiring no special knowledge of parallel programming. Our method rests on the observation that much of the appeal of high-level vectorized languages like IDL or MatLab stems from relatively simple internal loops over regular array structures, and that these loops are highly amenable to automatic parallelization with OpenMP. We discuss how ISIS, an open-source astrophysical analysis system embedding the S-Lang numerical language, was easily adapted to exploit this pattern. Drawing from a common astrophysical problem, model fitting, we present beneficial speedups for several machine and compiler configurations. These results complement our previous efforts with PVM, and together lead us to believe that ISIS is the only general purpose spectroscopy system in which such a range of parallelism – from single processors on multiple machines to multiple processors on single machines – has been demonstrated.

### 1. Problem: Underpowered Analysis

As noted in Noble et al (2006), parallel computation is barely used in astronomical analysis. For example, models in XSPEC (Arnaud 1996), the de facto standard X-ray spectral analysis tool, still run serially on my dual-CPU desktop. In this situation scientists tend to either turn away from models which are expensive to compute or just accept that they will run slowly. Analysis systems which do not embrace parallelism can process at most the workload of only 1 CPU, resulting in a dramatic  $1/n$  underutilization of resources as more CPU cores are added. At the same time, however, astronomers are well versed in scripting, particularly with very high-level, array-oriented numerical packages like IDL, PDL, and S-Lang, to name a few. They combine easy manipulation of mathematical structures of arbitrary dimension with most of the performance of compiled code, with the latter due largely to moving array traversals from the interpreted layer into lower-level code like this C fragment

```
case SLANG_TIMES:
    for (n = 0; n < $ na; n++)
        c[n] = a[n] * b[n];
```

which provides vectorized multiplication in S-Lang. This suggests that much of the strength and appeal of numerical scripting stems from relatively simple loops over regular structures. Having such loops in lower-level compiled codes also makes them ripe for parallelization with OpenMP on shared memory multiprocessors. Proponents contend that conceptual simplicity makes OpenMP more

approachable than other parallel programming models, e.g. message-passing in MPI or PVM, and emphasize the added benefit of allowing single bodies of code to be used for both serial and parallel execution. For instance, preceding the above loop with `#pragma omp parallel for` parallelizes the S-Lang multiplication operator; the pragma is simply ignored by a non-conforming compiler, resulting in a sequential program.

## 2. Parallelizing ISIS by way of OpenMP in SLIRP and S-Lang

ISIS (Houck, 2002) was conceived to support analysis of high-resolution Chandra X-Ray gratings spectra, then quickly grew into a general-purpose analysis system; it is essentially a superset of XSPEC, combining all of its models and more with the S-Lang scripting language, whose mathematical capabilities rival commercial packages such as MatLab or IDL. One of several distinguishing features possessed by ISIS is its ability to bring the aggregate power of workstation clusters, through a fault-tolerant PVM interface, to bear on general problems in spectroscopic analysis (Noble et al 2006). This paper complements that work by discussing two ways in which OpenMP has been used to enable shared-memory parallelism in ISIS. Like our PVM work, we believe this is another first for general-purpose X-ray spectroscopy, and is of added significance in that adapting ISIS for parallelism – both distributed and shared-memory – no modifications to its architecture or internal codebase were required.

The first manner in which ISIS was endowed with multicore capability involved loading a module of parallelized wrappers for C functions such as `exp` and `hypot` (Fig. 1). These bindings were created by SLIRP, which is distinguished by its ability to auto-generate vectorized wrappers such as

```
static void sl_atof (void)
{
    ...
    for (_viter=0; _viter < vs.num_iters; _viter++)
        retval[_viter] = atof((char*)arg1);
    ...
}
```

that allow functions which ordinarily accept only scalar inputs to also be used with array semantics. Adapting SLIRP to enable these vectorized wrappers to run in parallel was as simple as having it prefix the vectorization loop with an OpenMP `#pragma` as detailed above. This enables S-Lang intrinsics, all of which execute serially, to be replaced with parallel versions of the same name, transparently parallelizing the replaced functions. Our second multicore tactic involved minimizing the effects of Amdahl's law by parallelizing a number of S-Lang operators (and part of the `where()` function), through the addition of

```
#pragma omp parallel for if (size > omp_min_elements)
```

to the corresponding loops within the S-Lang interpreter source. The `if` clause in these directives was used to tune performance for small arrays, where the cost of threads outweighs the serial execution time. The speedup plots in Figs. 1 and 2 demonstrate significant performance gains from the use of shared-memory parallelism in ISIS. Our numbers represent measurements of prerelease GCC 4.2 -O2 builds on 2 CPUs running Debian Linux (Linux2) and Sun Studio 9 -xO3

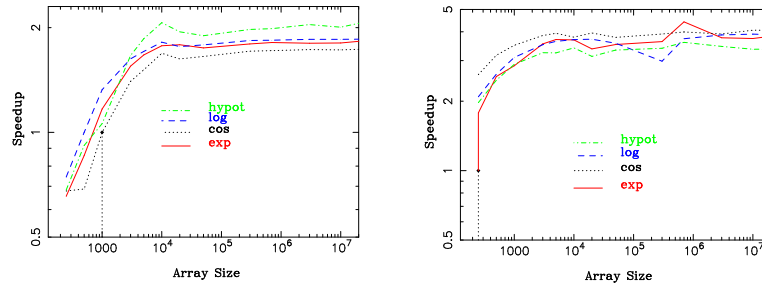


Figure 1. Speedups after replacing selected S-Lang math intrinsics with parallelized versions generated by SLIRP. Left: 2 CPUs on Linux (Linux2). Right: 4 CPUs on Solaris (Solaris4). The dotted vertical lines mark the inflection points where parallel performance begins to overtake serial.

builds on 4 CPUs running Solaris (Solaris4), with position independent compilation. Performance of the parallelized functions approaches the theoretical maximum of linear speedup as array sizes increase (Fig. 1), and the inflection points in the size of the arrays needed for nominal speedup from multithreading (represented by the dotted vertical lines) are relatively small, ca. 1000 elements on Linux2 and 250 elements on Solaris4.<sup>1</sup>

### 3. Case Study: Weibull Model in ISIS Spectroscopy

While ISIS supports custom user models in Fortran and C/C++, it can be faster to code them directly in S-Lang and avoid compilation steps during experimental tuning. In this section we discuss how one such function, a 4-parameter Weibull model coded for serial execution and taken directly from an active research project at MIT, parallelized using the techniques detailed above. Fig. 2 shows realized speedups converging on ca. 150% for Linux2 and 300% for Solaris4. These are sizable performance increases, and especially significant in that end-users need to do nothing – in terms of learning parallelism or recoding sequential algorithms – to obtain them; the same top-level model script can be used for both parallel and serial execution. Furthermore, recall that these models are used in the context of an iterative fitting process. Fits do not converge after just one iteration, and generating accurate confidence intervals – an absolute necessity for credible modeling – can require that tens or hundreds of thousands of fits be performed at each point on a parameter space grid. In such cases the speedups given here accumulate to significant differences in the overall runtime of an analysis sequence.

### 4. Conclusion: Transparently Parallel Scripting

We are witnessing the arrival of serious multiprocessing capability on the desktop: multicore chip designs are making it possible for general users to access

<sup>1</sup>More performance numbers and discussion, as well as the code for the Weibull function discussed herein, are given in the arXiv e-print at <http://arxiv.org/abs/0706.4048>

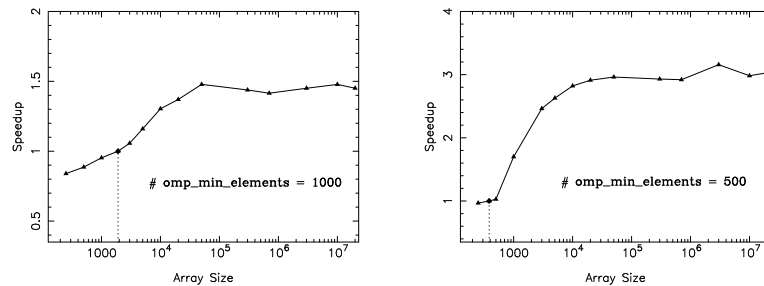


Figure 2. Aggregate speedup of the Weibull fit function due to the parallelized operators and functions discussed above. Left: Linux2, with inflection point at 1907 elements. Right: Solaris4, with inflection point at 384 elements.

many processors. At the granularity of the operating system it will be relatively easy to make use of these extra cores, say by assigning whole programs to separate CPUs. As noted with increasing frequency of late, though, it is not as straightforward to exploit this concurrency *within* individual desktop applications. In this paper we demonstrated how we have helped our research colleagues prepare for this eventuality. We have enhanced the vectorization capabilities of SLIRP, a module generator for the S-Lang numerical scripting language, so that wrappers may be annotated for automatic parallelization with OpenMP. This lets S-Lang intrinsic functions be replaced with parallelized versions, at runtime, without modifying a single line of internal S-Lang source. We have shown how S-Lang operators may also be parallelized with relative ease, by identifying key loops within the interpreter source, tagging them with OpenMP directives and recompiling. These simple adaptations, which did not require any changes to the ISIS architecture or codebase, have yielded beneficial speedups for computations actively used in astrophysical research, and allow the same numerical scripts to be used for both serial and parallel execution – minimizing two traditional barriers to the use of parallelism by non-specialists: learning how to program for concurrency and recasting sequential algorithms in parallel form. By transparently using OpenMP to effect greater multiprocessor utilization we gain the freedom to explore on the desktop more challenging problems that other researchers might avoid for their prohibitive cost of computation. The OpenMP support now available in GCC makes the techniques espoused here a viable option for many open source numerical packages, opening the door to wider adoption of parallel computing by general practitioners.

**Acknowledgments.** This work was supported by NASA through the Hydra AISRP grant NNG06GE58G, and by contract SV-1-61010 from the Smithsonian Institution.

## References

- Arnaud, K. A. 1996 in ASP Conf. Ser. 101, ADASS V, ed. G. H. Jacoby & J. Barnes (San Francisco: ASP)
- Houck, J. C. 2002, ISIS: The Interactive Spectral Interpretation System, High Resolution X-ray Spectroscopy with XMM-Newton and Chandra
- Noble, M. S., Houck, J. C., Davis, J. E., Young, A., Nowak, M. 2006, Using the Parallel Virtual Machine for Everyday Analysis, in ASP Conf. Ser. 351, ADASS XV, ed. C. Gabriel, C. Arviset, D. Ponz, & E. Solano (San Francisco: ASP), 481